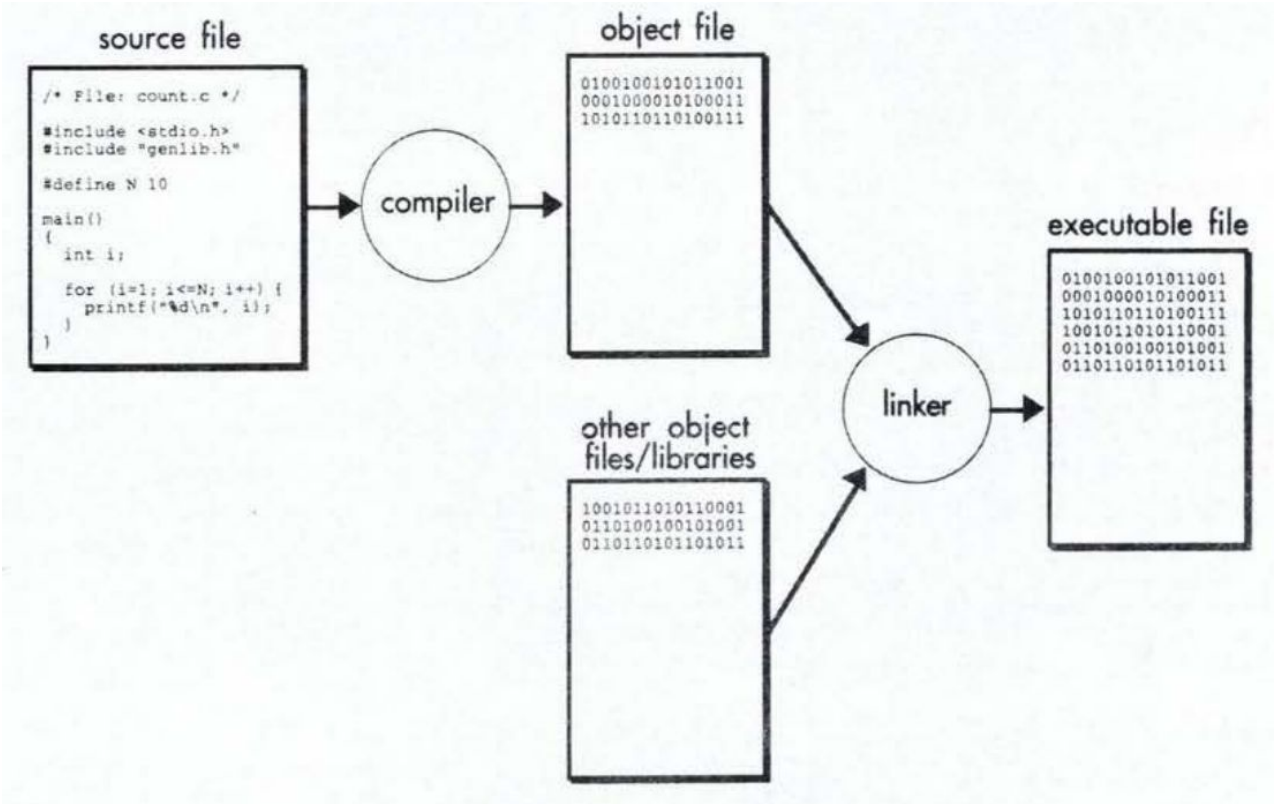
Estrutura de Dados I

Capítulo 1: Revisão de C

O que é C?

- Linguagem de máquina
- Linguagem assembly
- Linguagem de alto nível
- Arquivos:
  - Fonte
  - Pré-processado
  - Assembly
  - Objeto
  - Bibliotecas
  - Executável
- Processo de compilação e linking
- Portabilidade

O que é C?


source file
/* File: count.c */
#include <stdio.h>
#include "genlib.h"
#define N 10
main()
{
int i;
for (i=1; i<=N; i++) {
printf("%d\n", i);
}
}
compiler
object file
0100100101011001
0001000010100011
1010110110100111
other object files/libraries
1001011010110001
0110100100101001
0110110101101011
linker
executable file
0100100101011001
0001000010100011
1010110110100111
1001011010110001
0110100100101001
0110110101101011

Estrutura de um programa em C

- Instalação das bibliotecas necessárias
- Uso de Makefiles
- Uso de gccx ou gcc?
- Análise do código: potencia.c

Estrutura de um programa em C

- Comentários
- Bibliotecas
  - #include
  - < e >
  - “ e “
- Constantes simbólicas
  - #define
- Protótipos de funções
  - static
- Função main e printf
  - escape sequence
  - format codes

Variáveis, valores e tipos

- Declaração, atribuição e inicialização
- Propriedades das variáveis
  - Nome
  - Tipo
  - Tempo de vida
  - Escopo
  - Endereço
  - Valor
- Declaração de variáveis
  - tipo [name | namelist];

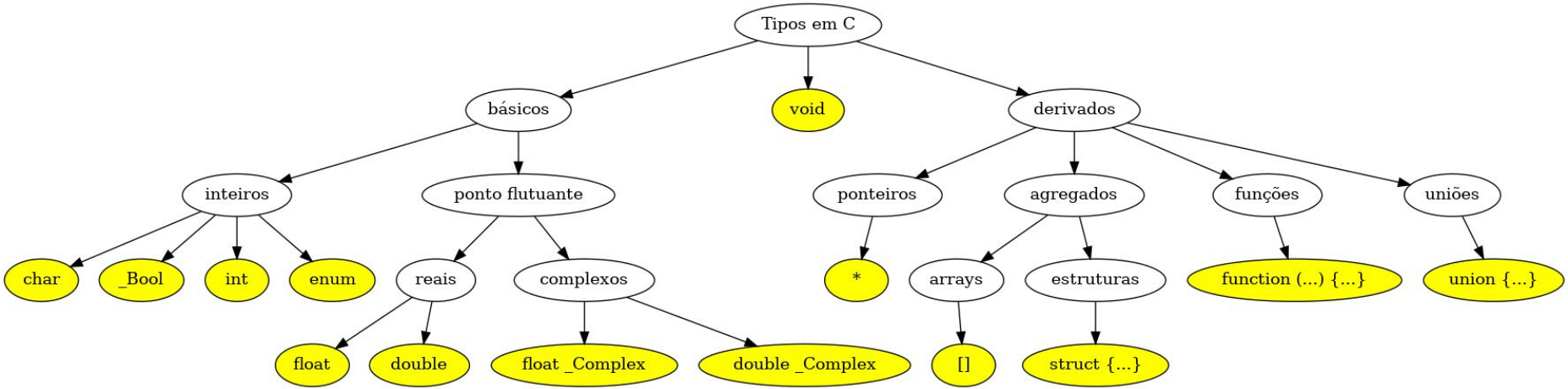
Variáveis, valores e tipos

- Identificadores
  - Regras
  - Keywords
  - Caso
  - Estilo de escrita
- Variáveis locais
  - Locais: escopo é dentro de um subprograma
    - Tempo de vida: enquanto o subprograma está ativo
  - Locais estáticas: escopo é dentro do subprograma, mas mantém valor entre as chamadas
    - Tempo de vida: enquanto o programa estiver em execução

Variáveis, valores e tipos

- Variáveis globais
  - Global: escopo é todo o arquivo e os outros arquivos através de extern
    - Tempo de vida: enquanto o programa estiver em execução
  - Global estática: escopo é todo o arquivo (nenhum outro arquivo acessa)
    - Tempo de vida: enquanto o programa estiver em execução
- Tipo de dado:
  - Conjunto de valores + Conjunto de operações

Variáveis, valores e tipos



Variáveis, valores e tipos

- Tipo int:

[signed] short int
[signed] int
[signed] long int
[signed] long long int

unsigned short int
unsigned int
unsigned long int
unsigned long long int

Variáveis, valores e tipos

- Tipo int:

$$(\text{short int}) \leq (\text{int}) \leq (\text{long int}) \leq (\text{long long int})$$

Variáveis, valores e tipos

- Tipo int:

```
signed short int:
    tamanho (bits): 16
    valor mínimo  : -32768
    valor máximo  : +32767

signed int:
    tamanho (bits): 32
    valor mínimo  : -2147483648
    valor máximo  : +2147483647

signed long int:
    tamanho (bits): 64
    valor mínimo  : -9223372036854775808
    valor máximo  : +9223372036854775807

signed long long int:
    tamanho (bits): 64
    valor mínimo  : -9223372036854775808
    valor máximo  : +9223372036854775807
```

```
unsigned short int:
    tamanho (bits): 16
    valor mínimo  : 0
    valor máximo  : 65535

unsigned int:
    tamanho (bits): 32
    valor mínimo  : 0
    valor máximo  : 4294967295

unsigned long int:
    tamanho (bits): 64
    valor mínimo  : 0
    valor máximo  : 18446744073709551615

unsigned long long int:
    tamanho (bits): 64
    valor mínimo  : 0
    valor máximo  : 18446744073709551615
```

Variáveis, valores e tipos

- Tipo int:
  - Literais inteiros
    - 10 = 10
    - 010 = OCTAL
    - 0x10CaFe = HEXADECIMAL (x ou X)
    - 0b1010 = BINÁRIO (b ou B)
    - L ou l = long
    - LL ou ll = long long
    - U ou u= unsigned
    - Pode-se misturar L, LL e U

Variáveis, valores e tipos

- Tipo ponto flutuante:

$$(\text{float}) \leq (\text{double}) \leq (\text{long double})$$

float
double
long double

```
Tamanhos (em bits) dos tipos reais:
      float:  32
     double:  64
long double: 128
```

Variáveis, valores e tipos

- Tipo ponto flutuante:
  - Literais de ponto flutuante:
    - sem sufixo = double
    - F ou f = float
    - L ou l = long double
    - E ou e = notação científica

Variáveis, valores e tipos

- Tipos para texto:
  - char:
    - Inteiros
    - ASCII
    - Aspas simples

	0	1	2	3	4	5	6	7	8	9
0	\000	\001	\002	\003	\004	\005	\006	\a	\b	\t
10	\n	\v	\f	\r	\016	\017	\020	\021	\022	\023
20	\024	\025	\026	\027	\030	\031	\032	\033	\034	\035
30	\036	\037	*space*	!	"	#	$	%	&	'
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	[	\	]	^	_	`	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	{	|	}	~	\177		

Variáveis, valores e tipos

- Tipos para texto:
  - char:
    - escape sequence

`'\a'`	The alert character (the terminal beeps)
`'\b'`	Backspace
`'\f'`	Formfeed (starts a new page)
`'\n'`	Newline
`'\r'`	Return (returns to the beginning of the line without advancing)
`'\t'`	Tab
`'\v'`	Vertical tab
`'\\'`	The character \ itself
`'\''`	The character ' (the backslash is required only in single characters)
`'\"'`	The character " (the backslash is required only in strings)
`'\`*ddd*`'`	The character whose ASCII code is the octal (base 8) number *ddd*
`'\x`*dd*`'`	The character whose ASCII code is the hex (base 16) number *dd*
`'\0'`	The null character (with zero as its character code)

Variáveis, valores e tipos

- Tipos para texto:
  - string:
    - NÃO EXISTE EM C!
    - As bibliotecas genlib.h e cs50.h definem para facilitar o estudo
    - Aspas duplas

Variáveis, valores e tipos

- Tipos para booleanos:
  - bool:
    - Em C existe o tipo _Bool
    - Header stdbool.h define “bool”, “true”, “false”
    - As bibliotecas genlib.h e cs50.h definem para facilitar o estudo
    - genlib.h define também TRUE e FALSE
    - 0 = falso
    - != 0 é verdadeiro (qualquer coisa diferente de zero é verdadeiro)

Input e Output

- C não define nenhuma funcionalidade de I/O, tudo deve ser feito através de bibliotecas especiais.
  - stdio.h é a padrão
  - simpio.h é a I/O simplificada:
    - GetInteger
    - GetLong
    - GetReal
    - GetLine
  - cs50.h é a da CS50:
    - get_char get_int get_long get_long_long
    - get_float get_double
    - get_string
  - simpio.h não tem prompt; cs50.h tem prompt

Input e Output

- Função printf é usada para output:
  - 1º argumento: string de formato (format string, control string)
  - Outros argumentos: valores dos especificadores de formato (format code, format specifiers)

Input e Output

`%d`, `%hd`, `%ld` — These formats display the value as a decimal number and are used with values of type `int`, `short`, and `long`, respectively. The `%` in the format code can be followed by a number representing the minimum field width. If the number is too short to fill the entire field, extra space is added at the left so that all numbers in that column will line up on the right.

`%f` — This format is used with values of type `float` or `double` and displays them as a number containing a decimal point. The `%f` format code may include a field width as in the `%d` format and may also include an indication of the desired precision, which is separated from the field width by a decimal point. In `%f` format, the precision indicates how many digits should be displayed to the right of the decimal point. Thus, if you use the format `%7.3f`, a number will be displayed in a seven-character field with three digits after the decimal point.

`%g` — This format is used with values of type `float` or `double` and is similar to the `%f` format as long as the number fits in a small space. Numbers whose magnitude is either very large or very small, such as 6300000.0 or .0000007, can be represented more compactly by displaying them in scientific notation, in which case they appear as 6.3e+6 or 7.0e–7. The `%g` format code may include a field width and precision as in the `%f` format, although the precision in `%g` format specifies the number of significant digits instead of the number of digits to the right of the decimal point.

`%c` — This format is used with values of type `char` and displays a single character.

`%s` — This format displays a string on the screen, character by character, and is used with arguments of type `string`. Percent signs appearing within the displayed string have no special effect. The format `%s` allows a field width and a precision like the numeric formats. For strings, the field width is usually preceded by a negative sign, which means that the field is aligned on the left side of the field rather than the right. The precision in the `%s` format specifies the maximum number of characters to be displayed.

`%%` — The `%%` specification is not really a format code but instead provides a way to display a single percent sign as part of the output.

Expressões

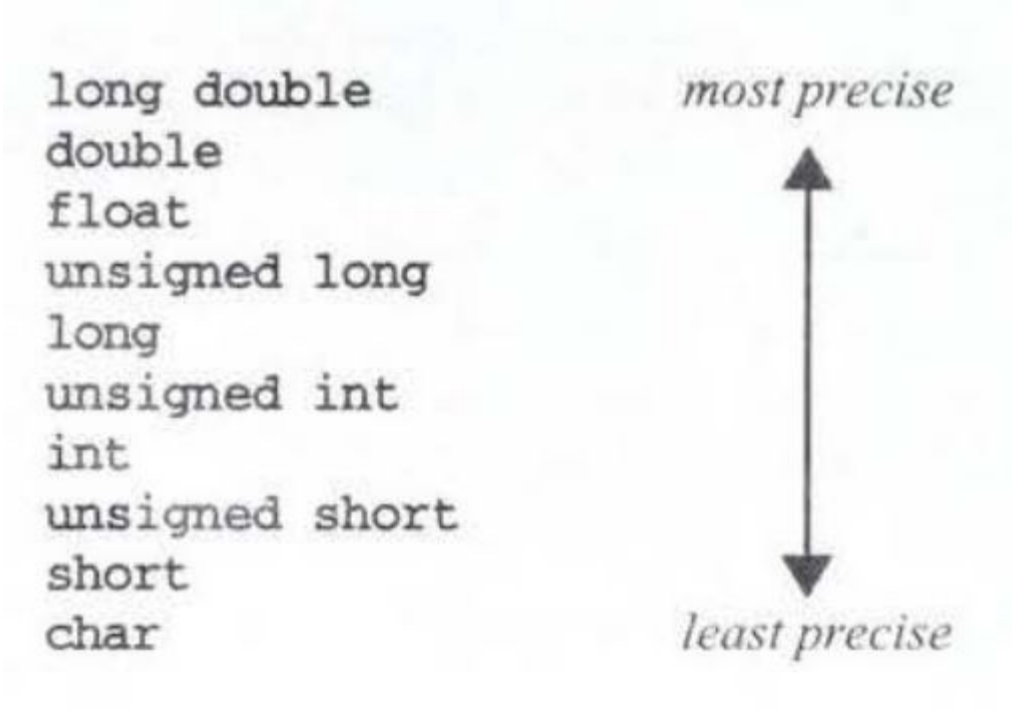
- Formadas por TERMOS e por OPERADORES
  - Termos: representam valores: constantes, variáveis ou chamadas de funções
  - Operadores: indicam uma operação computacional
    - Unário, binário ou ternário
  - Regras de PRECEDÊNCIA e de ASSOCIATIVIDADE
    - Verifica precedência; se mesma precedência, verifica associatividade
    - Associatividade pode ser à direita ou esquerda

Expressões

OPERATOR	DESCRIPTION	EXAMPLE	SIDE EFFECTS	ASSOC	PR
	Identifiers	`amount`	N	N/A	16
	Constants	`3.14159`			
	Parenthetical Expressions	`(a + b)`			
`[ ]`	Array Index	`ary[i]`	N	Left-Right	16
`f(...)`	Function Call	`doIt(x, y)`	Y		
`.`	Direct Member Selection	`str.mem`	N		
`->`	Indirect Member Selection	`ptr->mem`	N		
`++ --`	Postfix Increment • Decrement	`a++`	Y		
`++ --`	Prefix Increment • Decrement	`++a`	Y	Right-Left	15
`sizeof`	Size in Bytes	`sizeof(int)`	N		
`~`	Ones Complement	`-a`	N		
`!`	Not	`!a`	N		
`+ -`	Plus • Minus	`+a`	N		
`&`	Address	`&a`	N		
`*`	Dereference/Indirection	`*ptr`	N		
`( )`	Type Cast	`(int)ptr`	N	Right-Left	14
`* / %`	Multiply • Divide • Modulus	`a * b`	N	Left-Right	13
`+ -`	Addition • Subtraction	`a + b`	N	Left-Right	12
`<< >>`	Bit Shift Left • Bit Shift Right	`a << 3`	N	Left-Right	11
`< <= > >=`	Comparison	`a < 5`	N	Left-Right	10
`== !=`	Equal • Not Equal	`a == b`	N	Left-Right	9
`&`	Bitwise And	`a & b`	N	Left-Right	8
`^`	Bitwise Exclusive Or	`a ^ b`	N	Left-Right	7
`|`	Bitwise Or	`a | b`	N	Left-Right	6
`&&`	Logical And	`a && b`	N	Left-Right	5
`| |`	Logical Or	`a | | b`	N	Left-Right	4
`? :`	Conditional	`a ? x : y`	N	Right-Left	3
`= += -=`	Assignment	`a = 5`	Y	Right-Left	2
`*= /= %=`		`a %= b`			
`>>= <<=`		`a &= c`			
`&= ^= |=`		`a |= d`			
`,`	Comma	`a, b, c`	N	Left-Right	1

Expressões

- Misturando tipos:
  - coerion
  - cast



Expressões

- Divisão inteira: resultado é TRUNCADO
- % é o resto da divisão
- Atribuição:
  - LHS x RHS = variáveis x expressões (valores)
  - Sentença de atribuição
  - Atribuição embutida
  - Atribuição múltipla

```
result = 1;
```

```
z = (x = 6) + (y = 7)
```

```
n1 = n2 = n3 = 0;
```

Expressões

- Atribuição simplificada:

Operador	Significado	Exemplo		
+=	soma e atribui	x += y	igual	x = x + y
-=	subtrai e atribui	x -= y	igual	x = x - y
*=	multiplica e atribui	x *= y	igual	x = x * y
/=	divide e atribui quociente	x /= y	igual	x = x / y
%=	divide e atribui resto	x %= y	igual	x = x % y
&=	E bit a bit e atribui	x &= y	igual	x = x & y
|=	OU bit a bit e atribui	x |= y	igual	x = x | y
^=	OU exclusivo e atribui	x ^= y	igual	x = x ^ y
<<=	desloca à esquerda e atribui	x <<= y	igual	x = x<< y
>>=	desloca à direita e atribui	x >>= y	igual	x = x>> y

Expressões

- Incremento e decremento:

Operador	Significado	Exemplo
++	incremento	++x ou x++
--	decremento	--x ou x--

- **++x** (pré-incremento): soma **+1** à variável *x* **antes** de utilizar seu valor.
- **x++** (pós-incremento): soma **+1** à variável *x* **depois** de utilizar seu valor.
- **--x** (pré-decremento): subtrai **–1** da variável *x* **antes** de utilizar seu valor.
- **x--** (pós-decremento): subtrai **–1** da variável *x* **depois** de utilizar seu valor.

Expressões

- Booleanas:
  - Relacionais
    - Só em tipos atômicos
  - Lógicas
    - Esquerda para direita
    - Curto circuito
  - ?:

==	Equal
!=	Not equal
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

(condition) ? exp_1 : exp_2

!	Logical *not* (`TRUE` if the following operand is `FALSE`)
&&	Logical *and* (`TRUE` if both operands are `TRUE`)
||	Logical *or* (`TRUE` if either or both operands are `TRUE`)

Sentenças

- Dois grandes tipos de sentenças:
  - SIMPLES: realiza alguma ação e termina com “ ; “
    - chama função
    - faz atribuição
    - incrementa variável
    - ; faz parte da sentença simples
      - É um TERMINADOR, NÃO É SEPARADOR
  - CONTROLE: altera a maneira na qual as outras sentenças são executadas
    - Tipicamente se aplica a uma única sentença
    - Usa-se blocos para indicar que se aplica a um conjunto de sentenças
    - Blocos estão entre chaves
    - Compilador trata bloco como uma única sentença
    - Pode ter declarações de variáveis
    - Não usar ; após blocos

Sentenças

- Sentença if
  - if (condição)
    {
        expressões;
    }
    else if (condição)
    {
        expressões;
    }
    else
    {
        expressões:
    }

Sentenças

- Sentença switch
  - e = expressão de controle
  - c1, c2, c3 = valores CONSTANTES ESCALARES
  - break = impede a avaliação das outras sentenças
  - Pode usar um return

```
case 1:
case 2:
   statements
   break;
```

```
switch (e) {
   case c1:
      statements
      break;
   case c2:
      statements
      break;
   more case clauses
   default:
      statements
      break;
}
```

Sentenças

- Sentença while

```
while (conditional-expression) {
    statements
}
```

```
while (TRUE) {
    Prompt user and read in a value.
    if (value == sentinel) break;
    Process the data value.
}
```

Sentenças

- Sentença do while
  - do
    {
        sentenças;
    }
    while (condição);

Sentenças

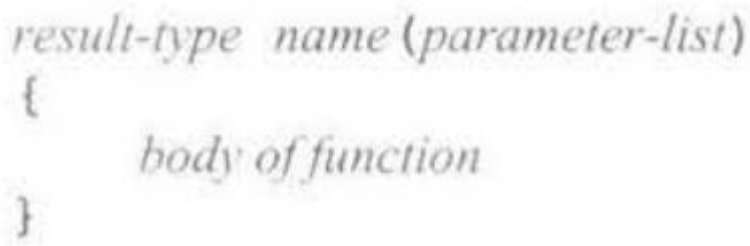
- Sentença for:

```
for (init; test; step) {
    statements
}
```

```
init;
while (test) {
    statements
    step;
}
```

Funções

- Conjunto de sentenças relacionadas que trabalham juntas e têm um único nome. Atenção:
  - Subprogramas: funções, predicados e procedimentos
  - Parâmetro x Argumento
  - Retorno x Efeito Colateral
  - Funções puras
  - Chamada da função
  - Assinatura
  - Cabeçalho
  - Protocolo



Funções

- Chamada de funções:
  - Chamador calcula valor de todos os argumentos
  - Sistema cria stack frame com espaço para todos os parâmetros e variáveis locais
  - O valor de cada argumento, em ordem, é COPIADO para o parâmetro correspondente
  - As sentenças são executadas até o return
  - O valor da expressão do return é avaliado e retornado como o valor da função
  - O stack frame é removido e todas as variáveis locais e parâmetros são descartados
  - O programa continua como valor retornado